УДК 004.272.43+004.272.32

Н.Л. Вереник[1], М.М. Татур[1], Е.Н. Сейткулов[2]
[1]Белорусский государственный университет информатики и радиоэлектроники, Беларусь
Республика Беларусь, 220013, г. Минск, ул. П. Бровки, 6
[2]Евразийский национальный университет имени Л.Н. Гумилева, Казахстан
Казахстан, 010008, г. Астана,ул. Мунайтпасова, 5

РЕАЛИЗАЦИЯ ГРАФОДИНАМИЧЕСКОЙ МАШИНЫ НА ВЫЧИСЛИТЕЛЬНОМ КЛАСТЕРЕ И ЕЕ ИНТЕГРАЦИЯ В ИНТЕЛЛЕКТУАЛЬНУЮ СИСТЕМУ

N.L. Verenik[1], M.M. Tatur[1], Y.N. Seitkulov[2]
[1]Belarusian State University of Informatics and Radioelectronics, Belarus
Belarus, 220013, Minsk, P. Browki Str., 6
[2]L.N. Gumilyov Eurasian National University, Kazakhstan
Kazakhstan, 010008, Astana, Munaitpasov Str., 5

GRAPH-DYNAMIC MACHINE IMPLEMENTATION ON COMPUTING CLUSTER AND ITS INTEGRATION IN INTELLIGENT SYSTEM

Н.Л. Вереник[1], М.М. Татур[1], Є.М. Сейткулов[2]
[1]Білоруський державний університет інформатики та радіоелектроніки, Білорусь
Республіка Білорусь, 220013, м.Мінськ, вул. П. Бровки, 6
[2]Евразійскій національний університет імені Л.Н. Гумільова, Казахстан
Казахстан, 010008, м Астана, вул. Мунайтпасова, 5

РЕАЛІЗАЦІЯ ГРАФОДИНАМІЧНОЇ МАШИНИ НА ОБЧИСЛЮВАЛЬНОМУ КЛАСТЕРІ ТА ЇЇ ІНТЕГРАЦІЯ В ІНТЕЛЕКТУАЛЬНУ СИСТЕМУ

В статье приводится описание оригинальной архитектуры абстрактной графодинамической машины (ГДМ) и процесс интеграции ГДМ в прикладную интеллектуальную систему. Рассмотрена реализация программной модели архитектуры на базе вычислительного кластера. Приведены результаты тестовых замеров производительности системы.

Ключевые слова: интеллектуальная система, семантическая сеть, параллельные вычисления, векторный процессор, ассоциативная память, графодинамическая машина.

In the article brief description for the original architecture of abstract graph-dynamic machine (GDM) and process of integration GDM in application system are given. Implementation of software model of the architecture on computing cluster is considered. The results of system performance test measurements are given.

Key words: intelligent system, semantic network, parallel computing, vector processor, associative memory, graph-dynamic machine.

У статті наводиться опис оригінальної архітектури абстрактної графодинамічної машини (ГДМ) і процес інтеграції ГДМ в прикладну інтелектуальну систему. Розглянута реалізація програмної моделі архітектури на базі обчислювального кластера. Наведено результати тестових замірів продуктивності системи.

Ключові слова: інтелектуальна система, семантична мережа, паралельні обчислення, векторний процесор, асоціативна пам'ять, графодинамічна машина.

Интеллектуальная система (ИС) – это программная или программно-аппаратная система, ориентированная на решение задач, традиционно относящихся к творческим и принадлежащих определенной предметной области. Совокупность знаний из этой предметной области составляет базу знаний,

которая является основным объектом форматирования, обработки и исследования. База разрабатывается для оперирования знаниями и, как правило, содержит вспомогательную служебную информацию, структурированную в зависимости от выбранного формата представления знаний.

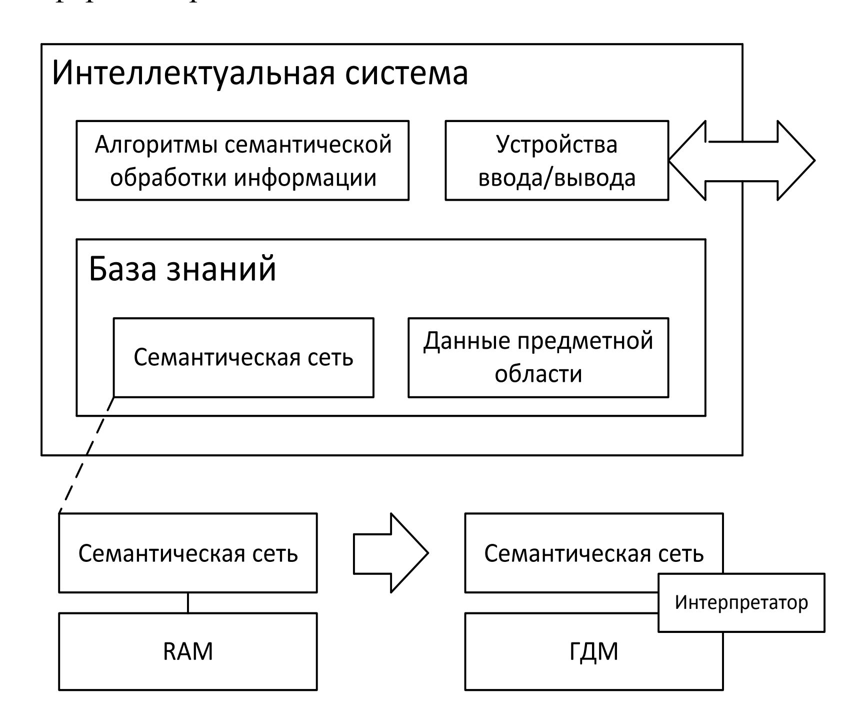


Рис.1. Схема интеграции ГДМ в интеллектуальную систему

На практике прикладная система включает в себя базу данных (БД) для хранения реально полезной информации (текстовые документы, изображения, мультимедиа, либо внешние ссылки на такого рода информацию), а также метаинформацию. Метаинформация используется для дополнительного описания хранимых данных, для придания им некоторой смысловой нагрузки в соответствии с конкретной прикладной задачей. Например, вводятся типы записей в БД, различные описательные атрибуты, логическая связь между данными, взаимозависимости и прочее. Совокупность всей метаинформации образует семантическую сеть, на базе которой и функционирует система [1] – выполняются сложные поисковые запросы и решаются другие интеллектуальные задачи. Аппарат семантических сетей не только обеспечивает визуализацию структуры информации, но, что важнее, позволяет разрабатывать формальные методы и алгоритмы анализа и модификации знаний.

К сожалению, на сегодняшний день не существует единого формата представления семантических сетей. В простейшем случае сеть может быть представлена связями между таблицами в реляционной БД. В более сложных случаях разрабатывается свой собственный формат хранения сети, построенный с учетом ряда допущений из-за тесной связи с конкретной задачей. Характерной чертой таких задач является их высокая структурная и динамическая сложность, решение в общем виде зачастую сводится к классу трансвычислительных – уже при относительно небольшом числе вершин графа (узлов семантической сети) время решения методом перебора превышает возможности любых теоретически мыслимых вычислительных систем (ВС). Однако с учетом специфики конкретных задач, а также в результате формулировки задачи на языке теории графов, можно

получить более простые методы решения, часто практически реализуемые. Известны попытки реализовать алгоритмы семантической обработки с применением серийных аппаратных платформ с параллельной архитектурой, таких как графические ускорители [2,3], кластеры и суперЭВМ [4,5,6].

Согласно ряду проведенных исследований [1] имеет смысл разделить верхний уровень ИС (т.е. методы семантической обработки информации) от ее нижнего уровня (уровня аппаратной реализации). Аппаратная платформа в таком случае может быть представлена в виде абстрактной графодинамической машины, ориентированной на анализ и обработку семантических сетей. Графодинамическая машина (ГДМ) – программная или аппаратная система, ориентированная на решение задач на графах. Согласно понятию графодинамики [7] входы, выходы и внутреннее состояние такой системы представляются структурами общего вида, принимающими значение из произвольного множества графов. Обработка информации в ГДМ трактуется как графодинамический процесс, т.е. процесс преобразования графовой структуры, в ходе которого меняется не только состояние элементов этой структуры, но и ее конфигурация (появляются и удаляются вершины, изменяются связи между ними).

В качестве конечной реализации такой абстрактной машины могут выступать любые ВС с любой подходящей архитектурой. Необходимую унификацию (т.е. платформенную независимость) обеспечивает программный интерпретатор, осуществляющий перекодировку программ с верхнего уровня на нижний (аппаратный) в рабочий формат ГДМ (рис.1).

Сложность, нерегулярность, иерархичность семантических графов приводит к проблемам в разработке программ для ГДМ. В настоящей работе предлагается один из подходов к построению унифицированной графодинамической машины на базе ассоциативной ВС (процессора ассоциативной памяти).

Алгоритмы преобразования графов

Основной задачей, решаемой интерпретатором, является регуляризация семантической сети для обработки на процессоре, т.е. формальная трансформация исходного графа семантической сети [1] к некоторому регулярному графу (рис. 2) [6]. Тогда регулярность графовой структуры позволит эффективно распараллелить алгоритмы вычислительного процесса ГДМ. Наряду с нерегулярностью структуры семантических сетей, отражающей информационные связи, определенную сложность представляет наличие большого числа типов элементов графа, необходимых для обеспечения разнообразия семантики представляемой информации.

Пусть имеется сематическая сеть $S(N,A)$ представляющая собой совокупность множества узлов семантической сети N и множества дуг семантической сети A. Главным отличием от классического графа является наличие у каждого узла и дуги набора разнообразных свойств и характеристик, а также возможность дуги входить не только в узел, но и в другую дугу. Необходимо получить из сети граф $G(V,E)$ регулярной структуры, без потери информации, хранимой в исходной сети. Для того чтобы не запутаться в обозначениях, при работе с исходной семантической сетью будем пользоваться терминами *узел* и *дуга*, а при работе с результирующим графом – терминами *вершина* и *ребро* (или ориентированное ребро).

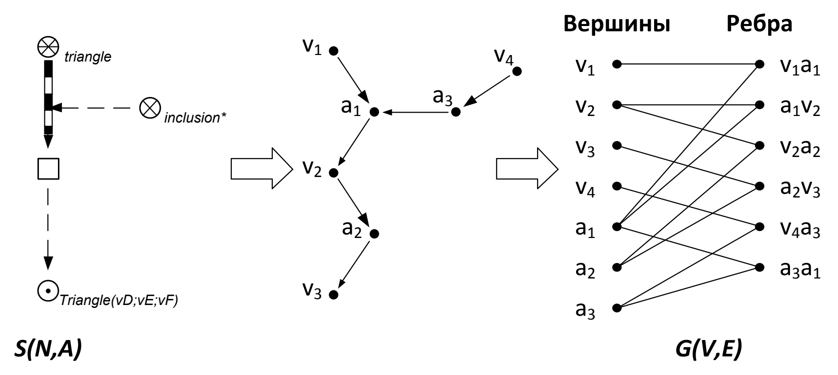


Рис. 2. Трансформация семантической сети в регулярный граф

Используемый алгоритм трансформации может быть представлен в виде псевдокода:

```
Для ∀n∈ N                                   (1)
    получим вершину v ← n                   (2)
    занесем v в V                           (3)
Для ∀uv∈ A,  u ∈ N,  v ∈ (N∪ A)             (4)
    получим вершину a ← uv                  (5)
    занесем a в V                           (6)
    занесем ua в E                          (7)
    занесем av в E                          (8)
```

Каждому узлу исходной сети *S* ставится в соответствие вершина графа *G*, каждой дуге сети *S* – вершина и два ориентированных ребра графа *G*. Полученные ребра соответствуют отношению, представляемому исходной дугой, и соединяют образованную вершину с двумя вершинами, инцидентными исходной дуге. Отметим, что на данном этапе вся семантика элементов исходной сети была перемещена в вершины графа *G*, а ребра содержат информацию о структуре исходной сети (связи и их ориентации).

Полученный таким образом двудольный граф *G* отличается своей регулярной структурой, позволяя свести алгоритмы семантической обработки данных к ряду несложных операций и обеспечить простоту распараллеливания вычислительного процесса.

Архитектура абстрактной ГДМ на базе ассоциативной ВС

Ассоциативная ВС есть не что иное, как одна из разновидностей параллельных вычислительных систем SIMD класса (рис. 3), в которых *n* процессорных элементов (ПЭ) представляют собой простые устройства, как правило, последовательной поразрядной обработки. При этом каждое слово (ячейка) ассоциативной памяти имеет свое собственное устройство обработки данных (сумматор). Операция осуществляется одновременно всеми *n* ПЭ. Все или часть ПЭ могут синхронно выполнять операции над всеми ячейками или над выбранным множеством слов ассоциативной памяти.

Время обработки *N m*-разрядных слов в ассоциативной ВС определяется выражением [7]:

$$T = mt(N/n + K),$$

где t – время цикла ассоциативной памяти; n – число ячеек ассоциативной системы; K – коэффициент сложности выполнения элементарной операции (количество последовательных шагов, каждый из которых связан с доступом к памяти). Таким образом, время обработки T является константой и в большей мере зависит от величины N/n, т.е. количества ячеек памяти, приходящихся на один ПЭ. После фиксации величины N/n (выбор конфигурации нитей GPU или настройка схемы ПЭ на FPGA) время обработки в идеальных условиях остается неизменным вне зависимости от общего объема обрабатываемой памяти.

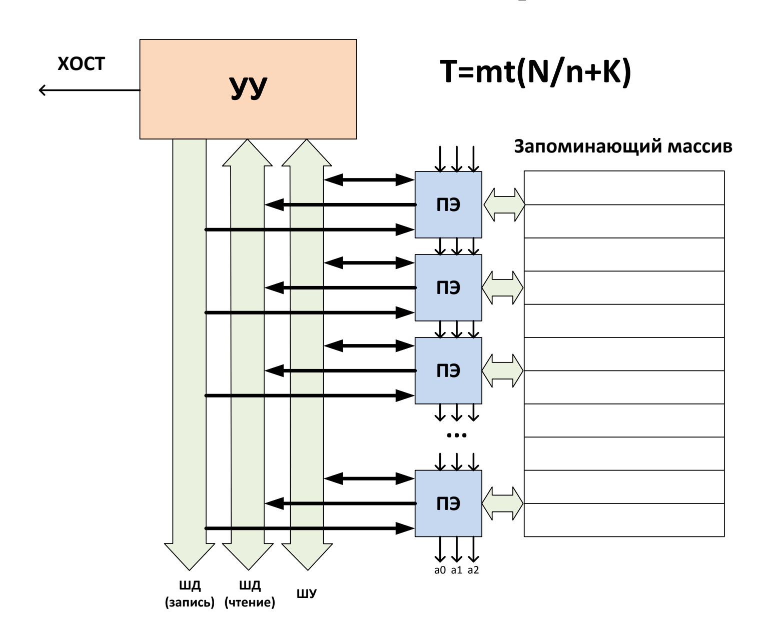


Рис. 3. Структурная схема ассоциативного процессора

Приведенная схема ГДМ представляет собой классическую SIMD архитектуру, состоящую из одного общего устройство управления (УУ) и линейки однотипных ПЭ со схемой, характерной для ассоциативной памяти (рис. 4). УУ обеспечивает последовательное выполнение команд исходной программы, в то время как ПЭ отвечают за непосредственную работу с памятью процессора (каждый ПЭ содержит локальную память). Коммуникация осуществляется посредством глобальных шин данных (ШД):

– ШД записи – шина, по которой исполняемая команда параллельно транслируется на все ПЭ;

– ШД чтения – шина, по которой выполняется последовательное чтение данных с процессорных элементов (ШД с временным разделением);

– шина управления (ШУ) – множество управляющих сигналов.

Каждый ПЭ представляет собой ассоциативное запоминающее устройство (АЗУ) и включает в себя (рис. 4):

– запоминающий массив для хранения N m-разрядных слов;

– регистр ассоциативного признака, куда помещается код искомой информации (признак поиска). Разрядность регистра k равна длине слова m;

– схемы совпадения, используемые для параллельного сравнения каждого бита всех хранимых слов с соответствующим битом признака поиска и выработки сигналов совпадения;

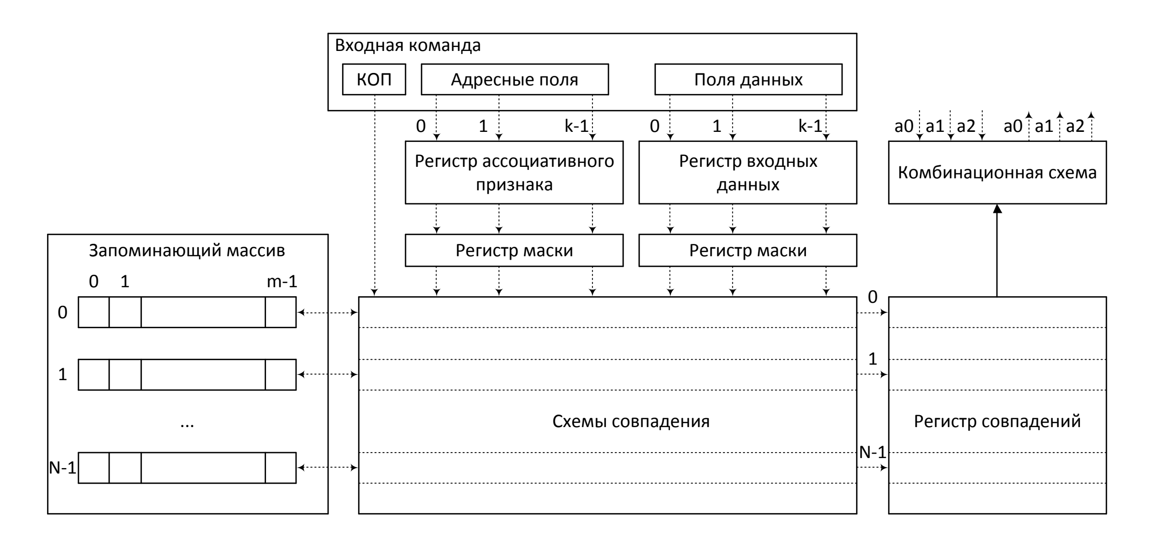


Рис. 4. Структурная схема ПЭ

– регистр совпадений, где каждой ячейке запоминающего массива соответствует один разряд, в который заносится единица, если все разряды соответствующей ячейки совпали с одноименными разрядами признака поиска;

– регистр маски, позволяющий запретить сравнение определенных битов;

– комбинационную схему, которая на основании анализа содержимого регистра совпадений формирует сигналы, характеризующие результаты поиска информации.

При обращении к ПЭ сначала в регистре маски обнуляются разряды, которые не должны учитываться при поиске информации (согласно полю S_M). Все разряды регистра совпадений устанавливаются в единичное состояние. После этого в регистр ассоциативного признака заносится код искомой информации (признак поиска, S_T) и начинается ее поиск, в процессе которого схемы совпадения одновременно сравнивают первый бит всех ячеек запоминающего массива с первым битом признака поиска. Те схемы, которые зафиксировали несовпадение, формируют сигнал, переводящий соответствующий бит регистра совпадений в нулевое состояние. Так же происходит процесс поиска и для остальных незамаскированных битов признака поиска. В итоге единицы сохраняются лишь в тех разрядах регистра совпадений, которые соответствуют ячейкам, где находится искомая информация. Конфигурация единиц в регистре совпадений используется в качестве адресов, по которым производится считывание из запоминающего массива.

Из-за того, что результаты поиска могут оказаться неоднозначными, содержимое регистра совпадений передаются на комбинационную схему, где формируются сигналы, извещающие о том, что искомая информация:

–*a0* – не найдена;

–*a1* – содержится в одной ячейке;

–*a2* – содержится более чем в одной ячейке.

Формирование содержимого регистра совпадений и сигналов *a0*, *a1*, *a2* носит название *операции контроля ассоциации*. Она является составной частью операций считывания и записи, хотя может иметь и самостоятельное значение.

Рассмотрим работу ячейки памяти классического АЗУ. При считывании сначала производится контроль ассоциации по аргументу поиска. Затем, при *a0=1* считывание отменяется из-за отсутствия искомой информации, при *a1=1* считывается слово, на которое указывает единица в регистре совпадений, а при *a2=1* сбрасывается маскируемая единица в регистре совпадений и извлекается

соответствующее ей слово. Повторяя эту операцию, можно последовательно считать все слова. Запись производится без указания конкретного адреса, в первую свободную ячейку. Для отыскания свободной ячейки выполняется операция считывания, в которой не замаскированы только служебные разряды, показывающие, как давно производилось обращение к данной ячейке, и свободной считается либо пустая ячейка, либо та, которая дольше всего не использовалась.



Рис. 5. Формат команды процессора

В обеих операциях производится контроль ассоциаций для выбора ячеек памяти, а также изменение состояния ячейки. В текущей реализации процессора обе операции реализованы посредством одной общей команды (рис. 5). Поиск ячеек определяется полями S_M и S_T, изменение состояния ячейки определяется полями W_M и W_D. W_M содержит маску, которая определяет разряды регистра, куда производится запись. W_D, в свою очередь, содержит значения, которые будут записаны в маскированные разряды. Запись может быть произведена как в первую выбранную ячейку, так и во все ячейки сразу в случае множественного совпадения. Первая выбранная ячейка будет определена с помощью аппаратно реализованной цепи очередности, позволяющей производить обращение к слову в порядке возрастания номера ячейки памяти независимо от величины ассоциативного признака. Выбор соответствующего режима осуществляется посредством поля «single cell» команды процессора.

Системный программист, таким образом, сам определяет как алгоритм чтения (в простейшем случае соответствующий классическому), так и алгоритм записи. Фактически, формат данных ячейки памяти является настраиваемым, и может изменяться в зависимости от решаемой прикладной задачи.

Формируемые на выходе ПЭ сигналы *a0*, *a1*, *a2* поступают на вход последующего ПЭ, который, в свою очередь, учитывает их при формировании своих выходных сигналов. Линейка ПЭ, тем самым, образует единую ассоциативную память, результирующие сигналы которой будут сформированы на выходе последнего ПЭ. В результате, процессор обладает хорошей масштабируемостью, позволяет наращивать объем данных системы простым подключением дополнительных ПЭ к общей линейке. Сохраняется основное преимущество ассоциативной памяти – увеличение количества ПЭ приводит к увеличению общего объема памяти без увеличения результирующего времени обработки данных.

Реализация ГДМ на вычислительном кластере

В рамках проекта представленная архитектура ГДМ была реализована на базе GPU (технология CUDA), с возможностью запуска как на одном графическом ускорителе, так и на кластере GPU. В дальнейшем планируется реализовать модель в виде проблемно-ориентированного процессора (на базе FPGA), что позволит сильно сократить итоговую стоимость одной ячейки памяти.

Общие технические характеристики ПО:

1. Программная модель реализована на базе вычислительного кластера, состоящего из управляющего узла и 7 вычислительных узлов. Характеристики узлов представлены в таблице ниже.

Управляющий узел	Вычислительный узел (x7)
• Blade: GPU SuperBlade SBI-7127RG • 2 x CPU Intel Xeon E5606 • 24 Gb RAM • 2x SSD 80Gb • 4x HDD 300Gb • InfiniBand 4x QDR (40Gbps) • Network 2x Gigabit Ethernet	• Blade: GPU SuperBlade SBI-7127RG • 2 x CPU Intel Xeon E5-2650 • 32 Gb RAM • 2x Tesla M2075 6 Gb RAM • InfiniBand 4x QDR (40Gbps) • Network 2x Gigabit Ethernet

2. Для работы с графическими процессорами на вычислительных узлах используется технология NVidia CUDA.

3. Для обмена данными между вычислительными узлами используется MPICH2 (стандарт интерфейса обмена данными MPI, Message Passing Interface).

4. Планирование задач и общее управление ресурсами вычислительного кластера осуществляется посредством TORQUE Resource Manager.

5. Основной рабочий проект реализован на C++, сборка проекта на тестовых машинах осуществляется в MVS2010, на кластере – посредством Makefile. Проект находится в открытом доступе на github.

Ниже приводятся результаты работы системы в тестовом режиме. Исполняемый тест включает в себя следующую последовательность действий:

1. Параллельная запись во все ячейки памяти фиксированного числового значения.

2. Поисковый запрос по тому же числовому значению.

3. Дальнейшая последовательная вычитка результата запроса (соответствует всему объему памяти процессора).

Конфигурацию процессора запишем в виде [A] (C; M; N), где A – количество вычислительных узлов, C – размер ячейки памяти (Бт), M – количество ячеек в одном ПЭ (размер локальной памяти ПЭ), N – общее количество ПЭ. Тесты были выполнены для различных конфигураций процессора согласно таблице:

Тест 1. Переменное число вычислительных узлов	Тест 2. Переменное число ПЭ	Тест 3. Переменное число ячеек в одном ПЭ
[1] (4; 1; 28627) [2] (4; 1; 57344) [3] (4; 1; 86016) [4] (4; 1; 114688) [5] (4; 1; 143360) [6] (4; 1; 172032)	[3] (4; 1; 86016) [3] (4; 1; 172032) [3] (4; 1; 258048) [3] (4; 1; 344064) [3] (4; 1; 430080) [3] (4; 1; 516096)	[3] (4; 1; 86016) [3] (4; 2; 86016) [3] (4; 4; 86016) [3] (4; 8; 86016) [3] (4; 16; 86016)

Было измерено среднее время выполнения инструкции, а также распределение временных затрат между операциями чтения, записи и временем простоя.

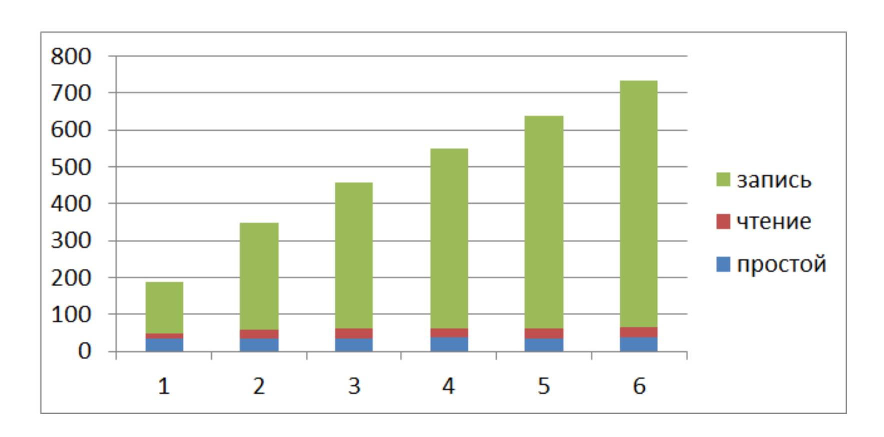


Рис. 6. Тест 1. Переменное число вычислительных узлов.
Среднее время выполнения инструкции, мкс

По результатам теста 1 видно (рис. 6), что подключение дополнительных вычислительных узлов на кластере (что соответствует подключению к процессору по 28627 ПЭ за каждый вычислительный узел) приводит к увеличению среднего времени выполнения инструкции. Это связано с текущей реализацией вычислительного процесса на кластере и будет улучшено в рамках работ по оптимизации программной модели.

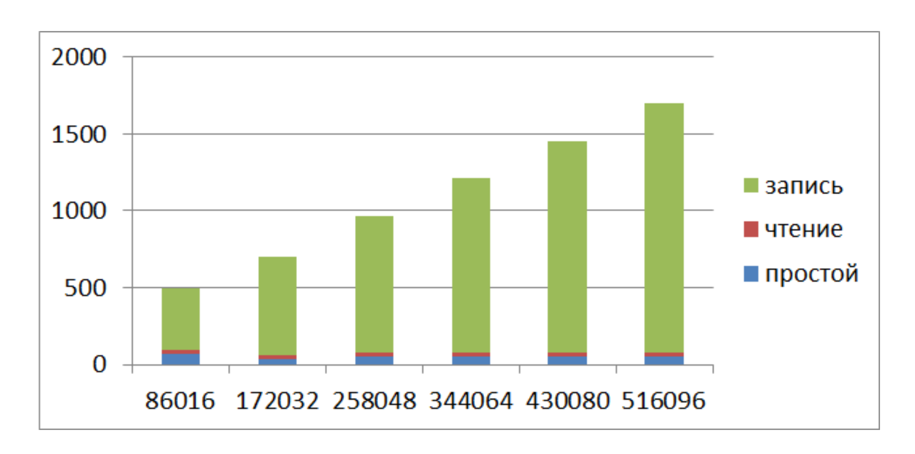


Рис. 7. Тест 2. Переменное число ПЭ.
Среднее время выполнения инструкции, мкс

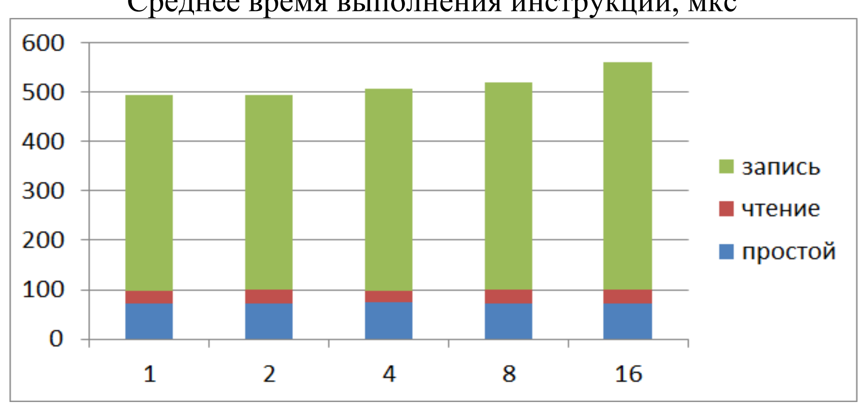


Рис. 8. Тест 3. Переменное число ячеек в одном ПЭ.
Среднее время выполнения инструкции, мкс

По результатам теста 2 видно (рис. 7), что увеличение количества ПЭ приходящихся на один вычислительный узел кластера, также приводит к увеличению

среднего времени выполнения инструкции. Это связано с тем, что каждый ПЭ эмулируется посредством отдельной нити GPU, количество которых ограничено некоторым верхним пределом. Соответственно, при превышении данного предела упадет общая производительность системы, т.к. выполнение операции на GPU займет несколько вычислительных циклов.

По результатам теста 3 видно (рис. 8), что увеличение количества ячеек памяти приходящихся на одну нить GPU (что соответствует увеличению локальной памяти ПЭ) практически не влияет на время выполнения инструкции. Это можно объяснить тем, что затраты на запуск ядра GPU значительно превышают затраты на выполнение каждой отдельной нити.

Заключение

В настоящей работе был изложен подход к построению прикладных интеллектуальных систем, основанный на принципе платформенной независимости верхнего уровня системы от уровня аппаратной реализации. В качестве апаратной платформы предложено использовать ГДМ, рассмотрен пример интеграции ГДМ в прикладную интеллектуальную систему. Формализован алгоритм преобразования семантической сети произвольного вида к графу регулярной структуры. Разработана архитектура абстрактной ГДМ SIMD-типа на базе ассоциативной вычислительной системы. Рассмотрены основные принципы функционирования процессора (ГДМ), формат данных и система команд процессора.

Для апробации результатов исследования была реализована программная модель предложенной архитектуры на базе вычислительного кластера (графические ускорители с NVidia CUDA в качестве вычислительных узлов). Разработано ПО для ускорения процесса интеграции модели в реальные прикладные системы.

В дальнейшем планируется продолжить исследование возможностей данной архитектуры (в частности программной модели на базе вычислительного кластера) посредством моделирования различных типовых задач семантической обработки, а также интеграции в существующие прикладные системы. Предстоит выполнить ряд оптимизаций ПО для увеличения производительности системы в целом.

Конечной целью исследования является построение аппаратного прототипа процессора на базе FPGA.

Литература

1. Голенков В. В. Графодинамические модели параллельной обработки знаний: принципы построения, реализации и проектирования / В. В. Голенков, Н. А. Гулякина // Открытые семантические технологии проектирования интеллектуальных систем: материалы II Междунар. научн.-техн. конф. (Минск, 16-18 февраля 2012 г.) – Минск : БГУИР, 2012. – С. 23–52.
2. André R. Brodtkorb, Trond R. Hagen, Martin L. Sætra. Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing, 73(1):4–13, January 2013.
3. Cristobal A. Navarro, Nancy Hitschfeld-Kahler, Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. Communications in Computational Physics, 15(2):285–329, February 2014.
4. Hiroaki Kitano, Dan Moldovan. Semantic network array processor as a massively parallel computing platform for high performance and large-scale natural language processing. In Proc. Int'l Conf. on Computational Linguistics (COLING '92), volume 2, pages 813–819, 1992.
5. Minhwa Chung, Dan Moldovan. Parallel natural language processing on a semantic network array processor. IEEE Transactions on Knowledge and Data Engineering, 7(3):391–405, June 1995.
6. Yahya Jan, Lech Jóźwiak. Scalable communication architectures for massively parallel hardware multi-processors Journal of Parallel and Distributed Computing. – November 2012. – Vol. 72. – Issue 11. – P. 1450–1463.

7. Айзерман М.А. Динамический подход к анализу структур, описываемых графами (основы графодинамики). I / М. А. Айзерман, Л.А. Гусев, С.В. Петров, И.М. Смирнова // Автомат. и телемех. – 1977. – № 7. – С. 135–151.
8. Вереник Н. Л. Разработка проблемно-ориентированных процессоров семантической обработки информации / Н. Л. Вереник, Е. Н. Сейткулов, М. М. Татур // Электроника инфо. – 2012. – № 8. – С. 95–98.
9. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2007. – 668 с.

Literatura

1. Vladimir V. Golenkov. Graphodynamical models of parallel knowledge processing / V. V. Golenkov, N. A. Guliakina // Open Semantic Technologies for Intelligent Systems (OSTIS-2012). – Minsk : BSUIR, 2012. – pp. 23–52.
2. André R. Brodtkorb, Trond R. Hagen, Martin L. Sætra. Graphics processing unit (GPU) programming strategies and trends in GPU computing. Journal of Parallel and Distributed Computing, 73(1):4–13, January 2013.
3. Cristobal A. Navarro, Nancy Hitschfeld-Kahler, Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. Communications in Computational Physics, 15(2):285–329, February 2014.
4. Hiroaki Kitano, Dan Moldovan. Semantic network array processor as a massively parallel computing platform for high performance and large-scale natural language processing. In Proc. Int'l Conf. on Computational Linguistics (COLING '92), volume 2, pages 813–819, 1992.
5. Minhwa Chung, Dan Moldovan. Parallel natural language processing on a semantic network array processor. IEEE Transactions on Knowledge and Data Engineering, 7(3):391–405, June 1995.
6. Yahya Jan, Lech Jóźwiak. Scalable communication architectures for massively parallel hardware multi-processors Journal of Parallel and Distributed Computing. – November 2012. – Vol. 72. – Issue 11. – P. 1450–1463.
7. Aizerman M. A., Gusev L. A., Petrov S. V., Smirnova I. M. A dynamic approach to analysis of structures represented as graphs (fundamentals of graph dynamics). I. Avtomat. i Telemekh. – 1977. – no. 7. – pp. 5–76.
8. Verenik N., Seitkulov Y., Tatur M. Development of ASIP for semantic information processing. Electronics info. – 2012. – № 8. – pp. 95–98.
9. Tsilker B., Orlov S. Organization of computers and systems: textbook for high schools. SPb.: Peter, 2007. – 688 p. (In Russian).

RESUME

N.L. Verenik, M.M. Tatur, Y.N. Seitkulov

Graph-dynamic machine implementation on computing cluster and its integration in intelligent system

In given article approach to build application intelligent systems using GDM (graph-dynamic machine) as a hardware platform is outlined. GDM is defined as a system oriented on solving problems on graphs. Inputs, outputs and internal state of such system are represented by structures of the general form, taking values in an arbitrary set of graphs.

Abstract GDM architecture of SIMD-class is developed which represents Content Addressable Parallel Processor (CAPP) in essence. Brief description is given for processor (GDM), processor's data format and instruction set, basic principles of functioning. Algorithm to transform arbitrary semantic network into a classic graph with regular structure is formalized based on which GDM operates and resulting in possibility to implement efficient algorithms of parallel processing. Example of GDM integration in application intelligent system is considered.

To approbate results of research the software model of proposed architecture was developed based on computing cluster and using NVidia CUDA technology. MPI technology is used to transmit information between cluster nodes. Software is developed

which is aimed at reducing expenses of GDM integration in real application systems. The results of the system performance test measurement are given; brief analysis of the received data is presented.

Н.Л. Вереник, М.М. Татур, Е.Н. Сейткулов

Реализация графодинамической машины на вычислительном кластере и ее интеграция в интеллектуальную систему

В данной статье изложен подход к построению прикладных интеллектуальных систем с использованием ГДМ (графодинамической машины) в качестве аппаратной платформы. Под ГДМ понимается система, ориентированная на решение задач на графах. Входы, выходы и внутреннее состояние такой системы представляются структурами общего вида, принимающими значение из произвольного множества графов.

Разработана архитектура абстрактной ГДМ SIMD-типа, представляющая собой ассоциативную ВС (вычислительную систему). Рассмотрены основные принципы функционирования процессора (ГДМ), формат данных и система команд процессора. Формализован алгоритм преобразования семантической сети произвольного вида к графу регулярной структуры, на основе которого функционирует ГДМ и в результате чего возможна реализация эффективных алгоритмов параллельной обработки. Рассмотрен пример интеграции ГДМ в прикладную интеллектуальную систему.

Для апробации результатов исследования реализована программная модель предложенной архитектуры на базе вычислительного кластера с использованием технологий NVidia CUDA. Для передачи информации между узлами кластера использовалась технология MPI. Разработано ПО, ориентированное на уменьшение затрат на интеграцию ГДМ в реальные прикладные системы. Приведены результаты тестовых замеров производительности системы, представлен краткий анализ полученных данных.

Поступила в редакцию 01.09.2015

